A Review of Middleware Approaches for Energy Management in Distributed Environments

Adel Noureddine, Romain Rouvoy, Lionel Seinturier

SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2012; **00**:1–30
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spe

A Review of Middleware Approaches for Energy Management in Distributed Environments

Adel Noureddine[1,2] and Romain Rouvoy[1,2] and Lionel Seinturier[1,2,3]

[1] *Inria Lille – Nord Europe*
[2] *University Lille 1 - LIFL CNRS UMR 8022, France*
[3] *Institut Universitaire de France*

SUMMARY

Energy management solutions and approaches for computer systems are becoming broadly available as energy concerns is becoming mainstream. Many approaches have been proposed to manage the energy consumption of the hardware, operating system, or software layers. The widespread usage of ubiquitous devices and the high coverage of networks (Wi-Fi, 3G) has led to a new generation of communicating and mobile devices that uses complex middleware platform functionalities. Therefore, energy management has emerged as a topic of research interest in the middleware layer and solutions specific to this layer are proposed along the more traditional ones existing at the other levels.
In this article, we report on a review of state-of-the art approaches for energy management middleware platforms. This article defines also an architectural taxonomy and compares existing approaches based on this taxonomy. In particular, we review middleware platforms and detail a number of approaches where energy management is handled. Finally, we review application scenarios where the energy management concepts at the middleware layer are applied in intelligent environments. Copyright © 2012 John Wiley & Sons, Ltd.

Received 27 January 2012

KEY WORDS: Energy-Aware Middleware; Autonomic Computing; Distributed Environments; Energy Management; Service-Oriented Architecture

1. INTRODUCTION

Reducing the energy consumption of connected devices and computers requires a comprehensive view of the different layers of the system. Sensors and actuators, used to monitor energy consumption and modify devices' options, need to be controlled by intelligent software. Applications running on the devices and the hardware itself also need to be monitored and controlled in order to achieve efficient energy savings. Many approaches have been proposed to manage the energy consumption of the hardware, operating system, network or software layers. However, with the widespread usage of ubiquitous devices and the high coverage of networks (Wi-Fi, 3G), a new generation of communicating and mobile devices is emerging. The energy consumption of this diversity of devices, and subsequently applications developed in different programming languages, is better managed through a layer capable of monitoring and managing both the hardware, operating system, network and application layers. Therefore, the middleware layer positions itself as a relevant candidate for hosting energy-aware approaches and solutions.

Many middleware platforms, architectures, optimization techniques and algorithms already exist for energy management of hardware and software. We therefore chose the reviewed approaches based on the priority given to energy management in the proposed solution. Our review on middleware approaches for energy management focuses on architectures and frameworks that emphasize on energy management in distributed environments. Only references and recent works

of the last years have been considered. Lots of efforts have been spent on energy management and optimization. Other approaches, such as energy optimization strategies and power consumption techniques, are not taken into consideration unless they target middleware platforms specifically or are applied on the middleware layer. We neither considered approaches where energy adaptation is limited to the hardware or software level without involving a middleware solution. Middleware approaches can adapt their core modules and/or the environment (*e.g.*, hardware, software) following energy objectives and we considered both approaches in our review. As most of nowadays distributed systems are connected with other applications and services, any viable solution should therefore incorporate solutions for energy awareness that emphasize and take advantage of the distributed nature of such systems. We selected the middleware approaches based on this criterion.

In this article, we report on a review on middleware approaches for energy management. To our knowledge, our review is the first to list and compare middleware approaches that specifically target the energy concerns. In Section 2 we review middleware approaches for energy management in distributed environments, proposing a detailed overview of the energy management issues in each approach. Section 3 overviews a number of approaches where a middleware layer is used for energy management of applications and devices in an intelligent environment. We compare the reviewed middleware approaches based on an energy taxonomy introduced in Section 4. Finally, we conclude in Section 5.

2. MIDDLEWARE APPROACHES FOR ENERGY MANAGEMENT

In this section, we review 12 middleware solutions targeted or specifically build for energy management. Middleware platforms provide abstraction of the underlying hardware, network, and operating system interfaces to the applications. S. Krakowiak defines middleware as [1]:

> In a distributed computing system, *middleware* is defined as the software layer that lies between the operating system and the applications on each site of the system.

The main goal of architectures and platforms for energy management is to optimize or reduce the energy consumption of hardware devices or software services. These approaches do not only optimize the energy consumption of applications and devices, but also optimize the consumption of the middleware platform itself. For a platform to manage energy efficiently, energy should not be considered as a non-functional requirement. It should rather be the core of the approach, eventually taking into account other requirements (quality of service, quality of context, user preferences, usability).

Many approaches integrating energy awareness or energy optimizations exist at the middleware layer. From the wide range of approaches, we select 12 middleware platforms responding to the following criteria:

1. *Middleware architectures or frameworks emphasizing on energy management in distributed environments.* We skipped middleware approaches that do not integrate the distributed dimension, or obviously energy or power management (thus excluding approaches limited to just monitoring or observing the energy consumption). The reason for limiting the study to distributed environments is that modern computer environments are massively distributed. With the democratization of cloud-computing, the widespread usage of connected mobile devices (*e.g.*, smartphones, tablets, laptops), and with the reduction of network costs for end-users, distributed usage scenarios are frequent. Managing energy of this rising usage is, thus, crucial for their success and market adoption.

2. *Recent work of the last years only.* Energy-aware approaches applied at different system layers have been proposed since the early days of computer science. Since 2005, more than 20,000 research papers related to energy management have been published [2]. Also, with the progress of technologies and the evolutions in customers' usages, approaches that were valid a decade ago may not offer today the same level of accuracy or energy saving as they

DOI: 10.1002/spe

were offering. Many approaches and energy related technologies became deprecated. The utilization of technologies and devices also change and evolve. This makes the need for new solutions a necessity (*e.g.*, the decline of the desktop PC and the rise of mobile devices and servers). With the high number of available publications, we argue that limiting the study to the most recent approaches and technologies allows us to provide a more representative view of the usable energy management approaches at the middleware layer.

3. *Limit review to approaches integrating the middleware layer.* The distributed nature of our environment requires energy management to be realized through a *connected layer*, *i.e.*, a layer capable to observe and control other layers involved in the distributed scenario. We argue that the middleware layer is best suited for this task as it observes and controls all software, operating system and hardware layers. We also limit our review to middleware approaches managing energy consumption, therefore we exclude energy monitoring approaches and energy measurement tools. Although monitoring the energy consumption of devices and software is important for efficient energy management, our review paper reports on middleware platforms where software or hardware adaptation is also achieved (thus approaches not limited to only monitoring and observing the energy consumption).

We select 12 middleware platforms integrating energy management approaches: Transhumance [3, 4, 5], Grace/2 [6, 7], CasCap [8], DYNAMO [9], PARM [10], ECOSystem [11], SANDMAN [12, 13], SleepServer [14], GreenUp [15] and the approaches reported in [16, 17, 18]. Table I and II summarize the positive and negative points of the middleware approaches for energy management. In addition, we overview 3 application middleware platforms where energy management techniques are used. These platforms manage energy for ambient applications and manage devices and software in intelligent environments (cf. Section 3).

2.1. Transhumance

Overview. Transhumance [3, 4, 5] is a power-aware middleware platform for data sharing on *Mobile Ad hoc Networks* (MANets). It supports collaborative applications and provides a set of communication facilities such as a publish-subscribe event system. Transhumance targets small networks (up to 20 nodes), moving at pedestrian speed (up to 5 km/h).

Architecture. The global architecture of Transhumance is composed of five functionality blocks (management components): energy, services, communications, groups and security. *Groups* defines and maintains communities of users who share a common interest. *Communications* manage communications inside a single node and between different nodes. *Services* provide advanced services to applications, such as replication service, consistency, and access rights. *Security* manages security of the previous three blocks. Finally, *energy management* adapts all previous blocks following the energy levels.

Energy Management. Each of Transhumance's functionalities is adaptable to the energy level. The energy management is policy driven with adaptation policies defining battery level thresholds at which adaptations are triggered. It follows a monitoring-decision-adaptation power management cycle. The energy management (cf. Figure 1) consists of two main elements: a *monitoring module* and a *resource manager*. The former monitors the local node and distant nodes for battery levels. The latter receives monitoring information and decides—using an adaptation policy—which adaptations to apply.

Adaptation policies (cf. Listing 1) follow the conditions/actions paradigm. Actions are defined as adaptations, which can be local on each node of the network, or global to affect all the system and multiple nodes. When the middleware platform adapts its behavior, the running applications may stop functioning or malfunction. Because the applications are affected by the system, they also need to be adapted to cope with the middleware platform adaptations. These adaptations are realized through applications' adaptation profiles (cf. Listing 2) that are sent to the middleware platform on

DOI: 10.1002/spe

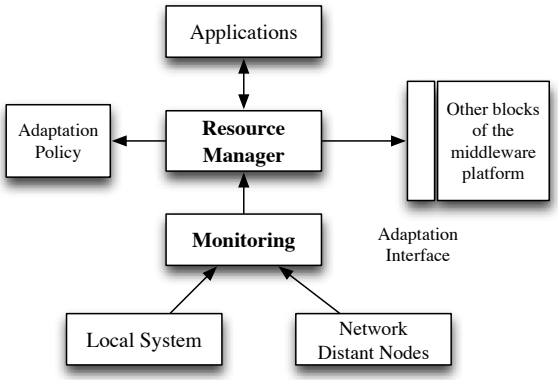

Figure 1. Transhumance's node energy management architecture. [5]

startup. When an energy level is triggered, the middleware platform notifies the applications, which adapt their behavior based on their adaptation profile and following the current energy level.

Algorithm 1: Adaptation Policy

if $(100 > LOCAL-ENERGY > 75)$ **then**
  No adaptation
end if
if $(75 > LOCAL-ENERGY > 45)$ **then**
  Neighborhood :: maximum-discovery-range = 4 hops
  Advertisement :: maximum-response-per-query = 25
end if
if $(100 > GLOBAL-ENERGY > 75)$ **then**
  No adaptation
end if
if $(75 > GLOBAL-ENERGY > 45)$ **then**
  Transport :: ciphering-enabled = FALSE
  Data :: lazy-propagation = TRUE
end if

Algorithm 2: Application Profile

if $(HIGH-ENERGY)$ **then**
  No adaptation
end if
if $(MEDIUM)$ **then**
  File-Transfer :: max-file-size = 2048 kB
  Transport :: ciphering-enabled = FALSE
end if
if (LOW) **then**
  File-Transfer :: max-file-size = 512 kB
end if
if $(VERY-LOW)$ **then**
  Transport :: no-ack = TRUE
end if

Figure 2. An example of adaptation policy and application profile used in Transhumance.

Examples. The authors validated their middleware platform using the Team Exploration treasure hunting game in a real life experiment. The game is played by a number of teams, composed of several members. Each member has a handheld device equipped with a Wi-Fi card. Using this device, players can access the map of the game area, and pictures that they must find from where they were taken. When a player proposes a location, all other team players must approve the proposal using the game interface on their device. When four images are located, team members must meet at the final meeting location to win the game.

This experiment shows the adaptive and collaborative nature of Transhumance. On the energy dimension, the authors compared the current consumption before and after adapting the transport protocol (specifically, with and without acknowledgments). As we can expect, energy consumption is lower when no acknowledgment is sent in the transport protocol. However, results show that, although acknowledgments represent 6% of the traffic, the energy consumption reduction is around 20%.

Discussion. Energy management in Transhumance focuses mostly on adapting the middleware platform's modules. Applications adaptations follow the middleware platform's own adaptations. As such, if the middleware platform does not adapt its modules in order to save energy (*e.g.*,

DOI: 10.1002/spe

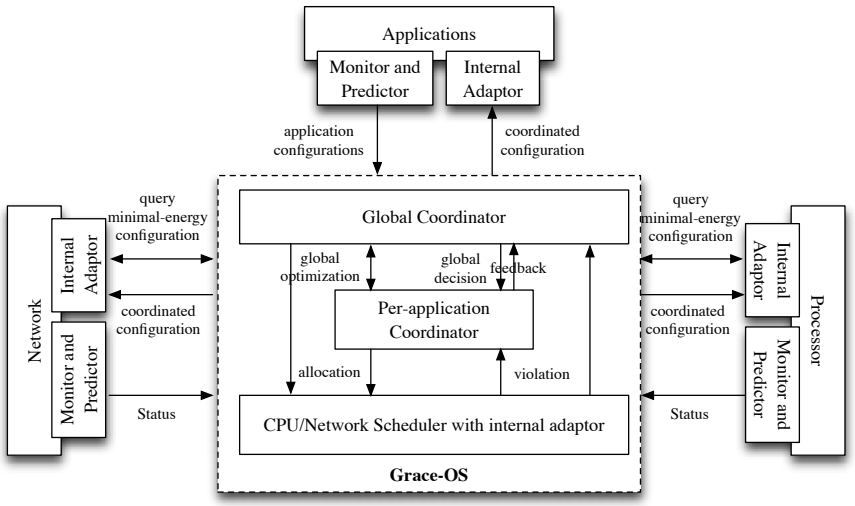

Figure 3. GRACE architecture. [6]

enable/disable messages encryption), then applications will not be adapted to the environment's changes.

Transhumance does not check for conflicts between applied actions. These is left for the user or administrator to guarantee that adaptations actions are not in conflict and does not damage the system integrity or are counterproductive (*e.g.*, wasting energy instead of saving it).

2.2. GRACE and GRACE-2

Overview. Global Resource Adaptation through CoopEration (GRACE) [6, 7] is a hierarchical adaptive framework for energy savings. It combines adaptations at different levels: *seldom and expensive global adaptation*, *frequent and cheap per-application adaptation*, and *internal adaptation on a single system layer*.

Architecture and Energy Management. GRACE uses a hierarchical approach (cf. Figure 3) that invokes expensive global adaptation (which considers all applications and all system layers), and inexpensive scoped adaptations (per-application adaptation where only one application is considered at a time), and internal adaptation where only a single system layer—but not necessarily one application—is considered. The global adaptation is applied occasionally and occurs at large system changes (*e.g.*, when an application enters or exits the system). The scoped adaptation is applied frequently and invoked when each job of the application starts or at a finer granularity (*e.g.*, every packet in the network). All adaptation levels are coupled with each other. This allows coordinated adaptations where the scope adaptation respects the resource allocation made by the global adaptation.

In the global adaptation, a global coordinator is responsible for resource allocation. For each configuration of the different system layers and applications, the coordinator determines its overall utility and resource usage. It then chooses the combination that maximizes the utility without exceeding the available resources. The global adaptation uses predictions of demands and availability when choosing the best configuration. These predictions are complex and therefore the global adaptation is invoked infrequently, only in response to large changes in the system.

The per-application adaptation is applied when an application job starts. At this moment, more accurate information is available about the available resources and the resource demand of each application's job. The adaptation goal is to find the configuration that will provide the utility expected by the global coordinator within its allocated CPU time and bandwidth while minimizing energy consumption. Unlike CPU time and network bandwidth, energy is a resource that can be

DOI: 10.1002/spe

conserved. Thus, the per-application adaptation tries to minimize the energy consumption while using up the time and bandwidth allocated.

Internal adaptation is used to redistribute the allocated time and bandwidth more optimally as application jobs may under run or overrun. It is also useful in response to variations in resource usage and availability that occur within a given job. And a system layer may apply an internal adaptation to determine its minimal energy configuration. This helps in reducing the configurations' search space used for the global and per-application adaptations.

Examples. The authors assessed their framework on several video and audio encoding and decoding workloads. The latter were run on four different scenarios of combination of CPU and network constraints. The results identify an high overhead of the global adaptation in comparison to the per-application adaptations (a factor up to 8).

The results also show that global adaptation achieves important energy savings compared to the non-adaptive base system. In non-constrained network scenarios, savings are of 9%, 35%, and 56% on average for global CPU, application, and CPU + application adaptations, respectively. For network-constrained scenarios, the average savings drops to 10%, 14%, and 27%.

Per-application adaptations (in both the CPU and the application itself), which represent the GRACE-2 framework, allow an additional savings of 6% on average compared to the global adaptation ones in non-constrained network scenarios, and an average of 27% in network-constrained scenarios.

Discussion. GRACE uses a hierarchical framework for energy-aware adaptations. This approach provides a flexible middleware platform that fits for distributed environments. The three layers system allows different granularity adaptations, from global and expensive adaptations to local per-application ones.

However, this approach requires a centralized global coordinator that needs resources-rich device to run on. This limits the benefits of the framework in environments where a large, but resource-poor, number of devices are present (*e.g.*, wireless sensor networks). The use of predictions in the global adaptation severely limits its frequency, thus limiting these adaptations to large changes in the system. This approach has also a drawback when the framework is used in a volatile environment. As applications and devices may appear or disappear frequently, the global adaptation is also invoked frequently, leading to more energy-consuming global adaptations (and ultimately to energy losses instead of savings).

2.3. Middleware for Energy-awareness in Mobile Devices

Overview. The authors propose an energy-aware middleware platform for mobile devices that is based on application classifications and power estimations to accomplish application-specific energy optimizations [16]. The middleware platform uses a policy manager to chose adaptive policies to apply on the system.

Architecture and Energy Management. The middleware platform is composed of six components:

i) a **resource manager** for handling multiple resource providers and requests for resources. This component relies on the system monitor to collect the runtime system information (such as the process list, processor frequency or the backlight luminance);

ii) a **machine learning application classifier** to support the recognition of new application types by analyzing the activity of applications (memory access mode and traffic characteristics). The classifier is first trained in order to be able to estimated application class using feature variables as input;

iii) a **power estimator** to estimate the power consumption of specific hardware components and applications during runtime. The estimator relies on hardware component-level power modeling for system-wide power estimation, and on application-level power modeling for application power estimation;

DOI: 10.1002/spe

iv) a **policy manager** that stores adaptive policies (written in semantic languages), and selecting policies that match request data (such as the application class and environment settings) sent by the processing engine;

v) a **messaging service** responsible for formatting, sending, receiving and parsing messages to/from remote entities;

vi) and a **processing engine** that schedules power adaptations automatically. It receives events from the resource manager and decides whether to request for application classification information from the application classifier, or request estimated power consumption from the power estimator, or request adaptive policies from the policy manager. It then applies the adaptive policies by invoking the corresponding adapters.

Examples. The authors evaluate their framework by running an adaptive version of mobile YouTube on their prototype. The prototype classifies the application based on monitoring data from the resource manager. It then estimates the power consumption and applies the adaptive policies if conditions are met (if the battery lifetime is low). Their experiments show an average of 8% energy consumption reduction when applying power adaptations using their framework.

Discussion. The major weakness of the architecture is the absence of any conflict resolving mechanism. Adaptation policies are added by the user/administrator and may conflict.

The approach also requires a training period for the application classifier to be anywhere effective. This may not fit very well in volatile environments where applications and hardware components change frequently. The dual power estimation (component hardware level and application level) may incur a non-negligible energy overhead in small sensor networks and very low capacity devices (embedded sensors and devices).

On the other hand, policy-based rules written in semantic languages allow very flexible adaptations. The approach can cover different environments with only having to modify policy rules.

2.4. CasCap

Overview. The authors propose CasCap [8], a framework for context-aware power management. The framework is based on three concepts: crowd-sourcing of context monitoring, functionality offloading and providing adaptations as services. Its architecture is composed of three components: mobile devices, Internet services, and clones. Mobile adaptations are based on adaptation policies that are also offered as services, while clones allow the mobile device to offload some processing to them for energy savings.

Architecture and Energy Management. CasCap is composed of three components:

i) **Mobile device** where energy savings take place. On mobile devices, the architecture of CasCap includes five components:

   a) a *Resource Manager* that collects resource consumption and useful information from sensors (such as the GPS receiver).

   b) a *Context Manager* responsible for generating (and download/uploading from/to the cloud) context information from the collected information by the resource manager.

   c) a *Scheduler* that adapts the mobile device based on the context and using adaptation policies.

   d) a *Policy Manager* responsible for installing, updating and sharing with other mobile devices the adaptation policies.

   e) and a *Communicator* for managing the wireless networking functionalities on the mobile devices.

DOI: 10.1002/spe

ii) **Internet services** responsible for storing and sharing context information between mobile devices and clones. These services also handle a crowd-sourced context monitoring service that collect and process context information from mobile devices and handles context queries. Finally, they offer adaptation services, such as transcoding or traffic shaping services for mobile devices.

iii) **Clones** are computer environment in the cloud that mobile devices can use to offload processing tasks, policy sharing, or scheduling of adaptations. Requests from mobile devices to internet services are therefore routed through clones. The use of clones is optional in CasCap.

Examples. The authors implemented a subset of CasCap architecture on a mobile device. They implemented a resource manager that monitors the WLAN interface (in particular, BSSID, signal and noise level) and the GPS location information. A context manager was also implemented and sends the WLAN and GPS collected information to the cloud. Adaptation policies were defined following the event-condition-action (ECA) principle. A TCP connection is used by the communicator for exchanging information between the mobile device and its clone. Preliminary results on a 4-minute YouTube video playback showed that when connecting through a WLAN access point with less users (thus higher SNR), energy consumption of network transmission is lower (at around 36% compared to another access point with higher number of users).

Discussion. The proposed architecture uses the cloud in order to propose adaptation services and functionality offloading. Adaptation as a service fits well in a ubiquitous environment where several devices may use a similar service. In this case, one implementation in the cloud is needed and allows energy savings by offloading all the processing to the cloud. However, the extensive usage of the cloud and internet services requires high usage of the network interface of mobile devices. The latter is one of the most energy expensive component in a mobile device. Therefore, optimizing the usage of the internet service by calculating processing/network costs tradeoffs is a most needed requirement for the architecture.

The weakness in the approach is more present in the experimentations rather then the theoretical architecture. No full implementation, neither any cloud-based service, were evaluated. Only minimal validations of trivial experiences were conducted (such as the WLAN energy consumption on a YouTube video). However, the authors identify where the major points of the architecture fits in the experiences. All what is left is now to validate these points with a global implementation of the CasCap architecture.

2.5. DYNAMO

Overview. DYNAMO [9] is a cross-layer framework for energy optimizations based on quality of service tradeoffs for video streaming in mobile devices. The framework uses a distributed middleware layer in order to adapt all levels of the system (*e.g.*, software, middleware, operating system, network and hardware). Its architecture uses a proxy server in order to perform *end-to-end* global and network adaptations with the mobile device (*e.g.*, dynamic video transcoding). *On-device* adaptations complement the *end-to-end* adaptions with local adaptations specific to the hardware and software of the mobile device (*e.g.*, LCD backlight intensity adaptation).

System model. DYNAMO's system model is composed of the mobile device and the proxy. The device's model has four system levels:

i) a **Hardware** level including the energy consuming components (*e.g.*, CPU, LCD, network card).

ii) an **Operating System** level that have access to the hardware through driver interface.

iii) a **Middleware** level that is composed by three abstract components:

   (a) a **system** component that resides within the OS.

DOI: 10.1002/spe

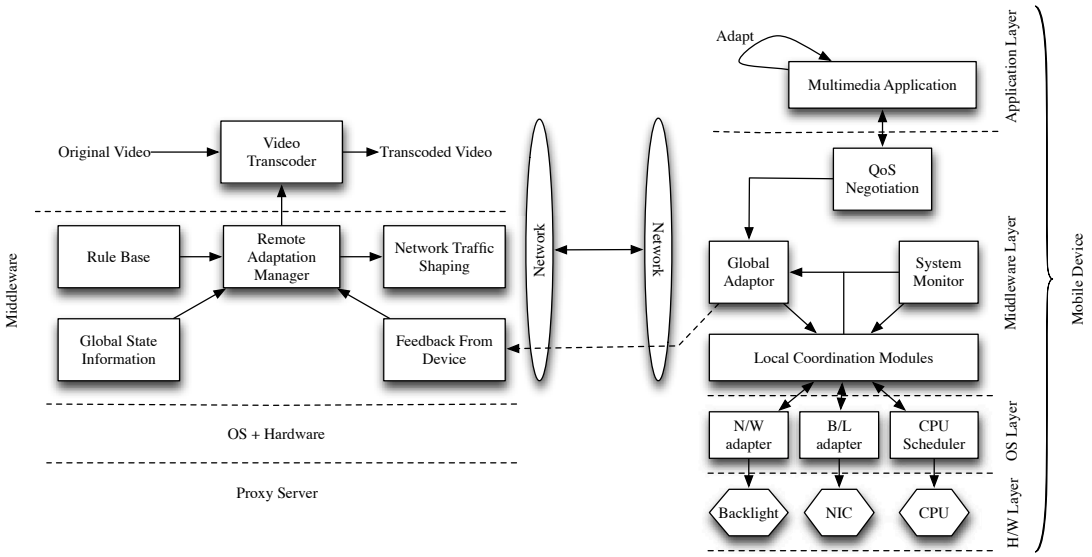

Figure 4. The End-to-end Cross-Layer Adaptation Framework of DYNAMO. [9]

(b) a **network** component for implementing communication protocols to talk with the proxy.

(c) and a **user level** component that performs adaptations using the information gathered from the user, OS and the network.

iv) and an **Application** level that provides user profiles and application context. It also dynamically negotiates quality of context with the middleware platform.

The middleware platform's modules installed on the mobile device can be *local state aware* (aware to local information such as CPU frequency or battery level) or *global state aware* (aware to information not available on the local device, such as network congestion or bandwidth availability). In the latter case, the middleware platform's modules installed in the proxy are responsible for determining these parameters. DYNAMO uses utility factors in order to achieve tradeoffs between performance and power consumption. On a video streaming example, the utility factor could be a measure of user satisfaction.

Architecture and Energy Management. The architecture of DYNAMO is distributed between the mobile device and the proxy (cf. Figure 4). On the proxy, a *remote adaptation manager* is responsible for providing quality parameters of the video stream (in order to maintain the highest system utility within the available energy in the device) using information collected from the device, the rule base and the end-to-end state (*e.g.*, network noise). It also informs the device about the remote changes allowing the mobile to adapt accordingly. On the mobile device, two main components manage adaptations: *i)* a *global adaptor* responsible for the reception and application of control information about remote adaptations; and *ii)* a *local coordinator* containing a set of OS and hardware access modules. These modules use information from the global adaptor to adapt the power consuming components.

Two main methods of adaptation are: end-to-end adaptations, and on-device cross-layer adaptations. End-to-end adaptations determine, using global knowledge of the system, whether the video request can be satisfied under the energy conditions of the mobile. In this case, the proxy has two tasks: energy-aware video transcoding, and intelligent network traffic shaping. Local adaptations, on the other hand, react to changes implemented by the proxy and to local application changes. Information is therefore exchanged periodically with the proxy (*e.g.*, LCD parameters, network adapter sleep times). Local adaptations can also perform quality of service negotiations with applications.

DOI: 10.1002/spe

Examples. The middleware platform is targeted to manage video streaming scenarios with respect to balance between power consumption and performance. A utility factor U_F was therefore defined for the system as the user satisfaction. U_F is equal to -1 if the system cannot stream the video while respecting the time, quality and power constraints. It is non-negative otherwise, following this definition:

$$U_F = \begin{Bmatrix} Q_{PLAY} - Q_a & IFF\ P_{VID} * T < E_{res}\ \& \\ & Q_{PLAY} \geqslant Q_a \\ -1 & Otherwise \end{Bmatrix} \quad (1)$$

where Q_{PLAY} is the streamed video quality, Q_a the threshold video quality level, E_{res} the residual energy on the device, T the duration of the video playback, and P_{VID} the average power consumption rate of the video playback.

The authors implement a prototype of DYNAMO running the video streaming scenario on a mobile device (iPAQ). Results show that energy savings with joint adaptations (*i.e.* of the different components, and cross-layers) can be up to 50% of the energy consumed by the three managed components (*e.g.*, CPU, network, display) without optimizations.

Discussion. The approach is built around the usage of a proxy in order to offload some energy-consuming functionalities from the mobile device. This approach is adapted to a network intensive scenario (such as video streaming) where the network overhead for communicating with a proxy is leveraged with the intensive use of the network for the video stream. Adapting the video stream on the proxy, thus allowing to reduce the streamed bandwidth and the required computation for processing it on the device, provides energy savings (up to 50% in the author's experimentation).

However, the proposed model and architecture will probably not perform as good as the paper's results in different scenarios. In a CPU intensive application, or a lighter network case (*e.g.*, web browsing), the energy gains are quickly overrun by the network overhead, and in particular in mobile devices (where the network card is in the top 3 of the energy consuming components).

On the other hand, the approach is based on utility functions. The reasoning for adaptation is based on the evaluation of this utility function, and the tradeoffs between energy consumption and the allowed variations of the quality of service parameters.

2.6. PARM

Overview. PARM [10] is a *Power-Aware Reconfigurable Middleware* for low-power devices. It dynamically reconfigures the component distribution on these devices and migrates components to proxy servers in order to save energy on the mobile client.

Architecture. Figure 5 depicts an example of a distributed system architecture in PARM. The system is composed of *distributed servers* (service providers, such as streaming video or web services), *proxy servers* (to host replicated data from servers), *meta-data repositories* (directory service, power broker), and *mobile clients*. The *directory service* stores the overall system state information.

The power broker is the central piece of the architecture. On large-scale systems, several power brokers will be distributed throughout the environment. It applies the adaptation decisions on the global system and/or on individual devices and determines the set of components to offload from mobile devices onto a proxy. It is also responsible for determining a new configuration for distributing middleware components between mobile devices and proxies. For that, it uses the current system state information (sent by the devices) and PARM policies (which define when and how often the broker reconfigures the devices).

Energy Management. The PARM framework is a flexible, reflective message oriented middleware platform. In addition to typical middleware platform components, it provides a set of additional independent components (*e.g.*, encryption/decryption, caching, clock synchronization) that are used for energy optimization.

Middleware services and components can be dynamically started/stopped/migrated. Components are migrated to a proxy in order to save energy on the mobile client. A component stub is left on the

Prepared using speauth.cls
DOI: 10.1002/spe

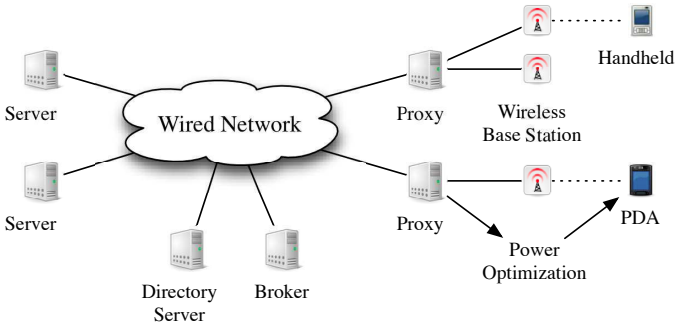

Figure 5. Example of a system architecture in PARM. [10]

client device when the component is migrated. The stub provides transparency to the application and handles communications with the remote migrated component. If a component cannot be migrated, it is either stopped, or its services degraded in order to save power.

PARM defines an algorithm to determine which components to migrate, and policies to determine when and how often the algorithm is executed. For that, the algorithm uses a parametric flow graph and determines the minimum-cut of the graph.

Examples. The authors simulated a model of the system and ran a serie of experimentation on it. Three classes of applications were used: *computation intensive* (class 1), *communication intensive* (class 2), and *computation and communication intensive* (class 3). Both sporadic-start and non-sporadic applications were used in these classes. Results show energy and service time gains (of about 15–45%) for class 1 applications when the reconfiguration took place every 5 minutes or less. Service time gains were less for class 2 applications (7–30%), and even lesser for class 3 applications. These latter shows loss when the reconfiguration was less frequent (once every 8 minutes or more). The results show that gains due to the PARM algorithm were the highest for computation intensive applications, but were moderate (and even negative) when communication intensive workload was added.

Discussion. The PARM algorithm has a worst case execution time of $O(n^3)$, which may lead to higher energy consumption of brokers and longer execution decision times when the environment is composed of a large number of mobile clients. The algorithmic approach using a flow graph is not the most effective approach in frequently changing environments. PARM goal is also limited to an environment where proxies, brokers and servers are abundant. It has one main core adaptation technique: component migration from mobile clients to proxy servers.

2.7. Green Computing: Energy Consumption Optimized Service Hosting

Overview. The authors propose a dispatch algorithm for data centers [17] in order to consolidate services dynamically into a subset of servers and temporarily shut down the remaining servers in order to save energy. The approach's goal is to minimize the number of running servers while still being able to respond to clients' requests and respect the QoS requirements described in SLAs.

Environment and Approach. The authors suppose that the environment includes three types of machines: *dispatchers* (that manage services requests and consolidation, and also when and how to shutdown and restart compute servers), *file servers* (that provide data sources) and *compute servers* (that host the services and a node manager responsible for monitoring the server and managing its shutdown). The environment offers different *Service Types* (ST) where each is uniquely identified by a service interface and a SLA. Compute servers can then be turned off if the remaining servers can meet the required QoS defined by all STs.

The dispatcher shuts down a server by sending a shutdown request to its node manager. The latter prepares the shutdown by making the server idle (refusing new service requests, waiting until all

Prepared using speauth.cls
DOI: 10.1002/spe

current service processing ends or migrate active sessions to another compute server), then sending the shutdown command to its operating system. Turning off the server can either be by a complete shutdown, hibernation or by suspending the system. The dispatcher chooses the optimal shutdown option based on the context of the environment.

Algorithm. The dispatcher has two main roles: *i)* selecting the compute server to handle incoming requests and *ii)* managing the pool of compute servers. In order to manage this pool and reduce the overall energy consumption, the authors first considered an optimization algorithm that they deemed not well suited. The search space for the algorithm can be large, thus being resource-hungry, and it is based on very specific assumptions (such as service requests are independent). The authors then proposed a probabilistic control algorithm. The principles of the algorithm are described as follows (quoting from article):

1. *If a running compute server s_i has idle time, the probability that service requests are dispatched to s_i is increased. Conversely, the probability that service requests are dispatched to the least energy-efficient compute server currently running s_r is decreased by the same amount. If the probability that service requests are dispatched to s_r becomes zero, the shutdown of s_r is initiated.*

2. *In an overload situation, the algorithm first tries to better distribute the load amongst the running compute servers, and restarts an extra machine only as a last remedy.*

Discussion. The approach has the advantage of respecting the SLA and QoS of client's requests while still trying to minimize the energy consumption. Although not rule-based or policy-based reasoning, the probabilistic algorithmic approach is not penalizing with regards to energy or time. Data centers are a resource-rich environment and the overhead of the algorithm should be minimal compared to the energy consumption of the servers.

However, the proposed approach and algorithm poses many assumptions, thus being limited to a small pool of case scenarios. It is also limited to data centers that host and handle services offered to clients. The authors identify several limitations of their algorithm, such as limited optimization criteria (only energy consumption and service response time are considered to date), centralized algorithm which may become a performance bottleneck, absence of fault tolerance and management of sudden fluctuations in service requests. These limitations are an ongoing research effort for the authors.

2.8. ECOSystem

Overview. Energy Centric Operating System (ECOSystem) [11] is a framework that manages energy consumption at the OS level. The framework is based on a new unit: *currentcy*, which is an abstraction of energy currency.

Model and Energy Management. Currentcy is a unified abstraction for the energy a system can spend on devices. A unit of currentcy represents the right to consume an amount of energy during a fixed amount of time. Using the currentcy definition, the framework (cf. Figure 6) manages energy following three orthogonal dimensions:

1. *Time*: a target lifetime is defined and divided into fixed-length epochs. At the beginning of each epoch, the allocation module generates an amount of currentcy available for all tasks. This amount depends on various parameters, such as the target lifetime or the remaining energy in the battery.

2. *Tasks*: management of the sharing of currencty among tasks. The allocation and scheduling modules cooperate to accomplish allocation of currentcy to tasks and providing opportunities (*e.g.*, access to devices) to spend the currentcy.

DOI: 10.1002/spe

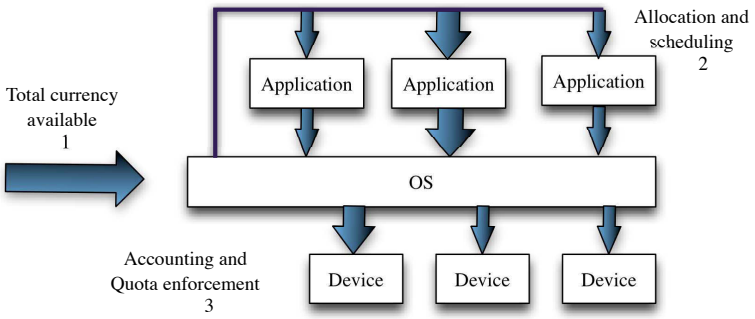

Figure 6. The ECOSystem framework. [11]

3. *Devices*: the actual energy consumers. When devices consume energy, the accounting module charges the tasks for the corresponding currentcy. If a task runs out of currentcy, then the system will block the task from accessing devices or consuming energy.

During each epoch, currentcy is limited, thus currentcy is allocated to tasks based on their importance. The latter is calculated using a proportional sharing approach. Tasks' currentcy can be accumulated form one epoch to another, however with a maximum cap. Without proper scheduling, tasks cannot consume their allocated currentcy. This leads the system to reallocate the exceeding currentcy to other tasks.

Examples. The advantages of the currentcy model and the energy management framework are highlighted in a comparison of three scheduling policies: *pay-as-you-go* (standard device schedulers), *static-priority* (scheduling priority is based on device's share value), and *currentcy-centric* (scheduler dynamically adjust scheduling priority based on the ratio of device's consumed currentcy to its entitled currentcy).

The authors ran gqview and ijpeg with equal shares. This experiment involves only the CPU scheduler. Results show that currentcy-centric policy is the only one to allow gqview to consume its allowed share of energy and to achieve its minimum delay.

Another experiment with both CPU and network schedulers showed similar results. With three applications competing for resources (RealPlayer, Netscape and ijpeg), only the energy-centric CPU and network schedulers allowed a proportional consumption of resources to the applications need.

Discussion. The authors propose an energy currency model, *currentcy*, and an OS framework for managing energy using this model. The currentcy model is a step forward toward unified energy management. The model, inspired from human's financial transactions, allows a flexible and generic approach to energy allocation.

However, the model and framework does not reduce or optimize energy consumption. It is only limited to providing a modeling and architectural infrastructure for energy transactions and allocation, but without reducing the energy utilization *per se*. Nevertheless, proposing a currency model for energy helps in raising awareness on the price of energy consumption, not only for the end-user, but also for developers and system administrators.

2.9. SANDMAN

Overview. SANDMAN [12, 13] is an energy-efficient middleware platform built upon the BASE middleware platform [19]. BASE is a minimal communication middleware platform for pervasive computing based on peer-to-peer principles. It is structured as an extensible micro broker, and plugins are used to support different communication protocols.

Architecture and Energy Management. SANDMAN is built as several extensions to BASE, and rely on three main concepts: *i)* reducing data transfer energy consumption by selecting the most

DOI: 10.1002/spe

efficient communication protocol, *ii)* switching idle devices to low power mode (sleep) in order to save devices' energy during their idle time, *iii)* and allowing clients to select the most energy efficient service.

In order to switch devices to idle mode while still being discoverable by users, SANDMAN uses an self-adaptable discovery protocol that can handle deactivated devices. At startup time, each device operates autonomously and answers users' discovery requests directly. During the system operation, neighbor devices, with the same mobility pattern, form a cluster and elect a *Cluster Head* (CH). The CH will be responsible for collecting information about all services in its cluster and answers users' discovery requests on behalf of the devices of its cluster. This will allow devices in the cluster to switch to sleep mode while still being discoverable through the CH.

SANDMAN relies on a transition strategy with a fixed inactivity threshold in order to meet the challenge in deciding if a device is unused and thus can be deactivated. In addition, SANDMAN relies on several other techniques to help making this decision. Techniques vary from middleware platform interfaces to let applications specify that a device is in use; sessions to allow users to specify that they are using a specific service, thus SANDMAN will not deactivate devices offering this service; to synchronization times negotiations between a user and a server, allowing both to temporarily sleep and communicate at given times.

Examples. The authors validate their approach by evaluating it with the Network Emulation Toolkit [*]. They compared the impact of the variation of group sizes on the energy savings. When the size is low (single device), the devices consume more energy than without SANDMAN's adaptations (devices are rarely clustered and the overhead of messages due to clustering is higher than what is saved by the sleeping devices). However, for larger groups, SANDMAN adaptations show high energy savings, up to 60% per device. With higher device speed, energy savings are smaller than with slower speed. This is due to less stable clusters with higher mobility (and thus the need for more frequent re-clustering).

Discussion. The SANDMAN approach targets energy consumption of idle devices in a communicating network. It achieves this by grouping devices with similar mobility patterns in a cluster and temporarily migrating nodes' advertisement to the cluster head. The approach however, have two weaknesses: *i)* the cluster head will have to answer discovery requests on behalf of devices of its cluster, thus having to be awake all the time and consuming more energy. No energy-aware approach is specified in the election of the CH. This may lead to small battery devices being elected as a CH, which may also lead to a quicker failure of the CH. *ii)* the CHs act as bottlenecks in the platform. The failure of a CH causes the devices of its cluster to be temporarily indiscoverable.

On the other hand, SANDMAN manages energy when devices are idle. It does not optimize the energy consumption of running applications, thus making the approach limited to situations where devices are used for short periods of time, and in non-critical environments.

2.10. SleepServer

Overview. The authors propose *SleepServer* [14], a software approach allowing energy management in desktop PCs. The approach allows machines to migrate to low power sleep states while still allowing their network connectivity. This is done using virtual machine proxy servers and virtual LANs.

Architecture and Energy Management. The architecture of SleepServer is network-proxy based. One of or more SleepServers (SSR) are added to an enterprise network and handles the host computers (H). Each SSR can manage a subset of the computer hosts.

For each computer host, a virtual image is instantiated. The latter is responsible for maintaining the network presence of the host when it is in a sleep mode. An *SSR Controller* manages the virtual

[*] http://net.informatik.uni-stuttgart.de

DOI: 10.1002/spe

appliances (*e.g.*, creation and configuration), the communications between the host images and an *SSR Client* software installed on the computer hosts, and resource allocation and sharing among the images. This controller and the images access the hardware resources through a *Resource Multiplexer* that can be the operating system itself or a hypervisor/virtual machine monitor.

On hosts, the SSR Client connects to the SleepServer machine and specifies its networks parameters (*e.g.*, MAC and IP address, firewall configurations). It also sends the state of running applications on the host and open TCP/UDP ports to the SSR Controller. Thus, the controller can mimic the host, in particular using the specifies network parameters and the firewall configuration. When the host goes to sleep mode, the virtual image can therefore respond to incoming packets on its behalf. If a request is received that requires the host itself, the SSR controller wakes up the host and disable its virtual image.

In particular, when a host goes to sleep mode, its SSR Client sends a message to the SleepServer's SSR Controller with the stat transition information. The controller enables the virtual image for the host and reconfigures the layer-2 switches in the network. For that, it uses a combination of gratuitous ARPs and packets sent to the gateway in the subnet. When the host goes out of sleep mode, the SSR Client informs the SSR Controller which disables the virtual image. The client sends also gratuitous ARP messages and packets to the subnet gateway in order to reconfigure the network's switches.

Examples. The authors shows that their approach can help reducing the energy consumption of PCs of about 68% compared with no SleepServer. In detail, they calculated the power consumption of users' PCs over a one month period, the first two weeks without SleepServer and the second two weeks using SleepServer. They deployed their SleepServer prototypes in over thirty PCs (desktops and laptops) in their university's department, mixing new and older computers and Windows and Linux operating systems. Powermeter were used to collect the energy consumed by the computers. Results show that energy consumption dropped by 30% when using SleepServer and when users had to manually put their PCs to sleep. This number grows up to 54% (or a total of 68% when compared to no SleepServer at all) when PCs where automatically put to sleep after one hour of timeout.

Discussion. The approach of SleepServer is similar to SANDMAN's approach. Both use additional proxy machines (a cluster head in SANDMAN and a sleep server in SleepServer) to maintain availability and network presence of the host while it is in sleep mode. Thus, it has the same weakness: *i)* the sleep server becoming a bottleneck for the platform. Its failure may cause host machines to become unavailable until they are wake up again; *ii)* and the sleep server will have to be awake all the time therefore consuming energy (albeit the energy consumed are compensated by the energy saved of the system).

On the other hand, the usage of SleepServer still requires additional hardware investment (in the form of the sleep servers themselves). It only manages energy for idle devices, allowing energy savings if frequent or long periods of sleep time occurs. SleepServer itself does not offer energy optimizations to applications or devices, but rather to the overall functioning of connected network of computers.

The approach, nevertheless, allows a transparent and heterogeneous migration of hosts' availability and network presence. Virtual images offer great flexibility for grouping images into a smaller number of sleep servers, or migrating them again to another sleep server (in particular in case of a SSR failure). They also offer security and isolation between the different virtual images.

2.11. Sleepless in Seattle No Longer

Overview. The authors present a system aimed to preserve the network accessibility of user machines in an enterprise network while still allowing them to go to sleep, and therefore save energy [18]. The system is based on two components: a *sleep proxy* for each subnet, and a *sleep notifier* program that runs on the user machine. The latter alerts the proxy when the machine goes

DOI: 10.1002/spe

to sleep. The proxy then sends out ARP probes to make sure all traffic to the sleeping machine are redirected to it. It responds to some packets while ignoring the majority.

Architecture and Energy Management. The approach proposed requires a sleep proxy per subnet, and that users install a sleep notifier daemon on their machines. The sleep notifier is responsible for notifying the proxy when the machine is going to sleep. In particular, when the machine is about to go to sleep, the notifier broadcasts *sleep notification* packets with the list of the machine's listening TCP ports. The notifier program broadcast this message to all its subnet, as there is no need for it to know the identity of the proxy.

The sleep proxy receives the broadcast and identifies machines that are about to go to sleep. It adds the machine to its list of sleeping clients and remaps the network router (by sending an ARP probe) so that future packets for the sleeping machine are redirected to the proxy instead. The proxy monitors incoming traffic to the sleeping machine, responding to ARP requests or neighbor discovery packets while ignoring the rest. When it recognize a TCP SYN packet meant for the sleep machine, it awakes it by sending a Wake-on-LAN [20] packet.

Examples. The implementation of the approach was deployed on 51 clients regrouped in 6 wired subnets (so, 6 sleep proxies were used), and for several months. Results show that ignoring most of the packets by the proxy prevent unnecessary wake-ups (*i.e.*, a TCP packet destined to posts on which the sleeping machine was not listening). Power savings varies greatly between machines as some were less solicited than others (thus allowing them to go to sleep more often and for longer periods, which results in better energy savings). On average, the approach allowed a 20% power savings, with some machines going up to 80% and others with just few percentages. A startup delay was also observed for the first connection to a sleep server, this delay is nevertheless absent for the following transactions. CPU load rarely exceeds 5%, while network bandwidth was also low (around 250 Kbps in 90% of the cases).

Discussion. The main advantage of this approach (and other similar ones) is allowing machines to go to sleep while still keeping them available for user requests. The only investment needed for user machines is the installation of a sleep notifier daemon. The latter doesn't need to know the address of the proxy (it broadcast its notifications), thus making adding/removing machines easy to manage for the proxy. The usage of an independent proxy frees the user machines from additional loads incurred from managing sleeping machines.

However, the proxy is a single point of failure in the system. If it goes down, sleeping machines can't be woken up again because their network traffic is redirected to the proxy. In addition, the sleeping policies are left for users or administrators to implement. A machine will go to sleep by its own (or the user's) initiative. Therefore, the proposed system can't save energy if machines (through their OSs, applications or users) dont go to sleep at all.

2.12. GreenUp

Overview. The authors propose *GreenUp* [15], a software-only approach for providing high-availability to sleeping enterprise workstation machines. GreenUp allows any workstation machine to act as a proxy for other machines so the latter can go to sleep while preserving their presence in the network subnet. Therefore when a machine goes to sleep, another one starts acting as proxy for it. It maintains its network presence by responding to ARP requests and IPv6 neighbor-solicitation messages on behalf of the sleeping machine. The proxy wakes up the machine when a user tries to connect to it (when a TCP SYN message is received).

Design and Energy Management. GreenUp allows machine to remain highly available while still allowing them to go to sleep. Its design is based on three ideas: *i) distributed management* that uses randomization to spread management load of machines evenly over proxies, *ii) subnet state coordination* to disseminate and broadcast the state of other machines (*i.e.* sleeping, awake) in the

DOI: 10.1002/spe

subnet, and *iii)* the usage of *guardians* to protect against having no machine awake to act as a proxy, thus maintaining a minimum number of awake proxies.

In particular, distribute management techniques make every proxy periodically probes a random subset of machines. If the machine is awake or if it is asleep but managed by a proxy, the probing proxy receives a response (from the machine or its manager proxy). If it does not receive a response, the proxy starts managing the machine itself. In order to avoid having multiple proxies managing the same machine, a priority function is defined for a proxy to manage a machine (the hash of the concatenation of the proxy and the machine's MAC addresses). Each proxy broadcasts periodically a message stating its managed machines. When a proxy receives this messages, its compare it priority with the sender's and decides to stop managing the machine or to respond to the sender (so the latter can stop managing the machine). Finally, the proxy sends ARP probes to ensure that all packets meant for the sleeping machine are delivered to the proxy instead. The proxy therefore monitors both incoming TCP SYN's and outgoing SYN's (the proxy is still a workstation machine) destined for the sleeping machine.

Subnet state coordination technique is based on three elements: *i)* each machine periodically broadcasts its state, *ii)* if the machine is asleep, its proxy manager is responsible for the broadcasting, and *iii)* periodically, all machines are awakened and proxies drop from their managed list any machine that does not broadcast during a defined window of time (to detect if a machine has failed).

Finally, some machines are always kept awake. They are considered guardians, and ensure that at any time a minimum number of machines are awake to act as a proxy for the sleeping ones.

Examples. The authors implemented GreenUp in *C♯* and tested it on nearly a hundred machines over a period of seven months. GreenUp was able to wake machines in 99.4% of cases with only 5 cases of failures due in part to Wake-on-LAN [20] failures (0.3% of WoL attempts are failures). Therefore, the mechanism used in GreenUp to wake up machines (Wake-on-LAN) is considered a reliable one. The implementation also uses less than 1% of CPU for parsing packets for one managed machine, and it is estimated that it would be about 12-13% if managing 100 machines by a single proxy (29% for 300 machines). And the network utilization (for managing 100 machines in 1000 node subnet) is less than 90 Kb/s of upload bandwidth and 60 Kb/s of download bandwidth, which is negligible in enterprise LANs. Finally, GreenUp increases the sleep time of machines by 31% in their experiments, which amounts to 175 kWh per machine per year of saved energy (assuming a desktop PC idle power is 65 W).

Discussion. GreenUp offers a software-only approach where existing machine are used as proxies to manage each other's sleep and network presence. The advantage is the absence of any hardware investment or modification of existing software (unlike for example SleepServer), making deployment easy.

Another aspect is the absence of bottlenecks or central proxy servers. Because each machine can act as a proxy, the failure of a machine or even a proxy is quickly remediated by having another machine act as a proxy for the newly unmanaged ones. *Guardian* machines ensures that there is always a minimum number of proxies available, thus limiting a complete blackout of the system.

However, although the approach manages the network presence of sleeping machines, it does not include policies to determine when machines go to sleep. This is left for users or administrator. Thus, GreenUp hopes to *induces users to choose more aggressive sleep policies and thereby save energy*, while still allowing machine availability.

3. APPLICATIONS

In this section, we outline a number of middleware solutions targeting applications where the approaches adopted are build specifically for the requirements and challenges of ambient

DOI: 10.1002/spe

Middleware Approach for Energy Management	Pros	Cons
Rule-based approaches		
Transhumance	Uses the intuitive conditions/actions paradigm	Focus on adapting the middleware platform's modules
		No conflict management for adaptation policies
		Can only be applied on a limited environment
CasCap	Offloads functionalities to the cloud to relieve the mobile device	Network (energy consuming component) usage required
		No complete experimentation yet
Middleware for Energy-awareness in Mobile Devices	Usage of applications classification	No conflict management
	Based on semantic policy rules	Requires training periods
DYNAMO	Combines adaptations at different system levels	Fits well only in a network intensive scenario
	Vision of the global and local context for adaptations	Requires hardware investment in a proxy
Proxy-based approaches		
SANDMAN	Usage of an auto-adaptable discovery protocol	Cluster head consumes maximum energy
	Nodes can go to sleep and still be discoverable	Cluster heads act as bottlenecks
	Usage of various techniques for detecting if a device is unused or not	
SleepServer	PCs can go to sleep and still maintain network presence	Sleep servers act as bottlenecks
	Virtual Images allows flexibility and security	Sleep servers and state transitions consume energy
Sleepless in Seattle No Longer	PCs can go to sleep and still maintain network presence	Proxy server acts as bottlenecks and is a single point of failure
	A proxy frees machines from additional loads	Sleeping and energy policies left for users to define
GreenUp	PCs can go to sleep and still maintain network presence	Sleeping and energy policies left for users to define
	Software-only approach	
	Absence of bottlenecks or central server	

Table I. Summary of rule-based and proxy-based approaches

applications and devices in intelligent environments. Table III summarizes the positive and negative points of the architectural-based approaches.

3.1. AIM

Overview. The authors present a framework architecture for modeling, visualizing and managing energy consumption of home appliances [21]. The work is achieved under the AIM consortium [22]. The goal is to manage the energy consumption of appliances while providing this information to the user using communication devices.

DOI: 10.1002/spe

Middleware Approach for Energy Management	Pros	Cons
Other approaches		
GRACE/2	Combines adaptations at different system levels	Requires a centralized global coordinator
	Flexible approach: infrequent and expensive global adaptations, and frequent and cheap local adaptations	Limited advantages in volatile environments
PARM	Efficient component migration and redistribution for energy optimization	Worst case execution time of $O(n^3)$
		Necessity of proxies, brokers and servers presence
	Works well for computation intensive applications	Moderate or negative gains in communication intensive applications
Energy Consumption Optimized Service Hosting	Energy savings while respecting SLAs and QoS	Requires many assumptions, limited to data centers
	Minimizing the number of running servers	Centralized algorithm, limited optimization criteria, no fault tolerance
ECOSystem	Proposes a currency model to quantify energy management	No energy optimization *per se*
	Allocation and scheduling following Time, Tasks and Devices dimensions	

Table II. Summary of other middleware approaches

Architecture and Energy Management. AIM bridges the indoor and outdoor networks through the AIM system logic (cf. Figure 7), in order to provide control functions to users, network operators and energy generation utilities. Gateways can host part or all of the system logic, and manage communications, encryption and apply the centralized control logic.

Energy Management Devices (EMDs) manage the power logic and are controlled by the AIM gateway. EMD provides two types of power management logic: *i)* a power monitoring to collect energy consumption data from appliances and buffers them to the user and *ii)* and a power control for managing external communication interfaces and applying the user commands on the given appliances.

The AIM gateway is composed of three modules: *i)* a machine-to-machine interfaces module for providing a common API for implementing gateway-based services, *ii)* an identify management module for user authentication and identification, and *iii)* a services synthesis module for the creation of new composite services.

The user can control all appliances and devices manually. He can also define a set of user preferences that will enforce some specific behavior. However, AIM system uses user profiling in order to self-adapt to user habits. The profiling records events of user interaction with the home devices, then uses a learning algorithm to process the collected data and extract settings for the energy management system that meet user requirements. The goal of the profiling is to provide better system autonomy by replacing user manual interaction with automatic procedures performed on request.

Examples. AIM's functionality is intended to be used by three categories of users: *utility providers*, *operators* and the *local end-users*.

DOI: 10.1002/spe

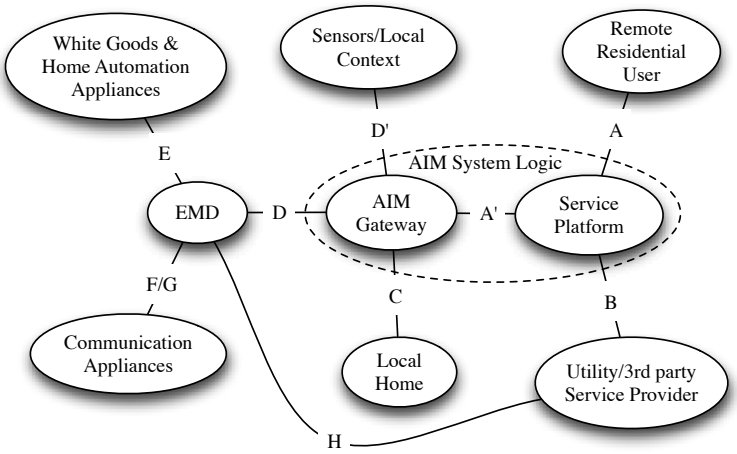

Figure 7. AIM system logic interfaces. [21]

A local end-user example is related to the energy management of home devices according to user preferences. For example, the system adjusts the bedroom, bathroom, and kitchen temperatures when the user wakes up in the morning. It lowers them during the day when he is outdoor.

Communication operators can provide services for users to access and monitor their energy consumption. The user installs EMDs and an AIM gateway, and subscribes to the operator's services. After that, the user can monitor his consumption by accessing the operator web portal. Advanced services are also offered, such as controlling devices or sending notifications to the user's mobile phone.

Utility providers can offer flexible cost model services to the user. The latter will receive each day a pricing profile for the next day. This profile includes energy prices for the day, per hour. Thus, the user's system can adjust energy consumption based on the price.

Discussion. The AIM project proposes solutions for energy management in smart homes while integrating services from network and communication operators and from power distribution operators. The convergence of these three actors (users, operators and utility providers) into a single approach, allows coordinated energy management based on multiple criteria (user preferences, electricity prices).

However, the proposed approach requires multiple devices (one or more EMDs, the AIM gateway) that should be connected and running all the time, thus it can't be sized to resource-poor environments (*e.g.*, Wireless Sensor Networks). The approach also requires processing and reasoning time, which is not always available on volatile or mobile pervasive environments.

3.2. Energy-awareness in the HYDRA middleware platform

Overview. The HYDRA middleware project [23, 24, 25] is co-funded under the Sixth Framework Programme (FP6). Its goal is to develop a middleware platform for networked embedded systems that allow developers to create ambient intelligence applications. The middleware platform uses the principles of *Service-Oriented Architecture* (SOA) and *Model-Driven Architecture* (MDA). One application example is a smart home integrating energy efficiency features.

Architecture. The architecture of the HYDRA middelware platform hides device-dependent and network-dependent details from the applications, and provide abstractions interfaces of the display, communication ports, input and memory management of each device. It also uses peer-to-peer technologies for identification and utilization of available services.

HYDRA considers each device, *Ambient Intelligence* (AmI) application or other subsystem as a unique web service, thus allowing interoperability at a semantic level. Therefore, devices can

DOI: 10.1002/spe

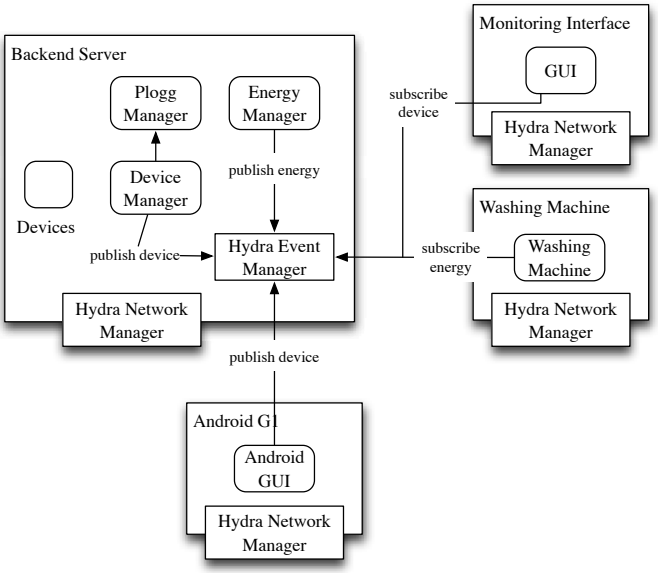

Figure 8. The energy-aware smart home system architecture. [24]

be controlled independently of the network protocol used (Wi-Fi, ZigBee, Bluetooth…). The network layer interconnects devices and people while the SOA approach in the service layer allows interoperability at a syntactic level.

Energy Management Example. An example of energy management using the HYDRA middleware platform is an energy-aware smart home [24]. The example application objective is to integrate energy efficiency into a smart home infrastructure, with the user having an interface for monitoring and controlling the environment. Wireless ploggs are attached to devices and used to monitor the energy consumption of these devices. The system uses five main managers (cf. Figure 8):

i) *Device and Plogg Management*, which administers information about devices and which plogg is connected to which device. It also regularly polls all ploggs for their current values of devices' energy consumption.

ii) *Energy Management*. This manager is an interface between the electricity producer and clients. It can, for example, manages the variation and changes of the electricity price.

iii) *Network Management* that abstracts the communication protocols.

iv) *Event Management*. This manager uses a publish/subscribe approach for event management. Two categories of events are implemented: device-related (*e.g.*, unplugged devices or ploggs does not deliver any more data) and energy-related (*e.g.*, changes in electricity price).

User interaction is also considered as an important factor. The authors used a mobile phone as a magic lens to offer an augmented reality experience. Users can use their phone's camera to point on a device and get information (*e.g.*, energy consumption in watt or in currency, such as in euro) and control options, in form of a video overlay.

Discussion. HYDRA is a generic middleware platform for heterogeneous devices. It facilitates communications between these devices and provides an architectural infrastructure for network abstraction, event management and services communications. It is not built for energy efficiency, nor it includes energy management techniques. It does, however, provide the infrastructure for construction energy-aware solutions. Thus, it relies solely on developers and constructors initiatives to propose energy-aware solutions.

DOI: 10.1002/spe

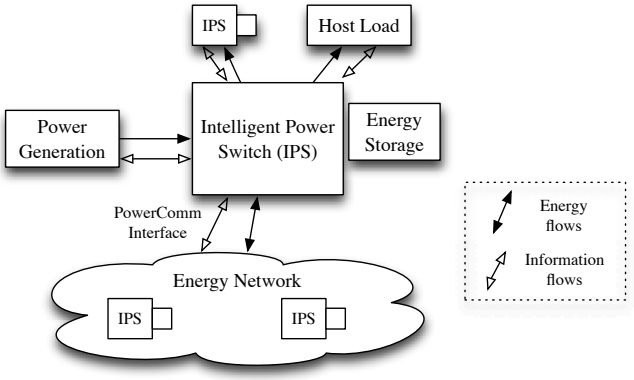

Figure 9. LoCal architecture schematic. [26]

3.3. *LoCal*

Overview. LoCal grid [26] is an intelligent and autonomous power system for local energy generation, distribution and sharing. It is a complementary architecture to the traditional electrical grid with objective of reducing the grid load. It allows the use of renewable energy sources and energy storage transparently by using an *Intelligent Power Switch* (IPS). This switch standardizes the electrical flow coming from different sources (with different characteristics, *e.g.*, voltage, frequency) and manages communications between the nodes of the grid. LoCal grid is developed by the LoCal research team [27] at the University of California, Berkeley.

Architecture and Energy Management. The global architecture of the LoCal grid is presented in Figure 9. The architecture is a distributed architecture following a peer-to-peer communication model. A communication system exists in parallel to the power system. Each node of the network is composed of an IPS, energy storage and power generation capabilities. The LoCal grid can operate independently or as an incremental overlay over the traditional grid.

The IPS is the core block of the grid. It provides abstraction of the underlying implementations to the nodes, and standardizes energy interaction characteristics (*e.g.*, voltage or power levels). The power management system (responsible for monitoring and moderating power loads, generation and external power flows) can be integrated in an IPS (or can be a separate component).

Discussion. The architecture solidifies the traditional grid by providing local power generation and storage capabilities. It also allows a more resistant grid as a failure in one LoCal node will not affect the overall grid. However, the architecture does not include any energy optimization or reduction approaches. The IPS could be an ideal candidate for integrating energy-aware power distribution, generation and storage, with a goal of reducing the overall energy consumption of the connected clients. It could also add strategies for advantaging green power generation or storage components.

4. COMPARISON AND DISCUSSIONS

In order to compare the solutions reported in this review, we defined a taxonomy to describe the various properties of each middleware platform solution. Figure 10 summarizes this taxonomy, while table IV synthesizes a general comparison of the middleware approaches for energy management presented in this review.

DOI: 10.1002/spe

Applications' Middleware	Pros	Cons
AIM	Converges users, operators and utility provides together for coordinated energy management	Difficult to apply on resource constrained environments
	Usage of context learning and user profiling for self-adaptation	
HYDRA	Provides infrastructure for constructing energy-aware scenarios	No energy optimizations *per se*
	Follows SOA principles and considers devices and applications as web services	
LoCal	Solidifies the traditional power grid	No energy optimizations *per se*

Table III. Summary of smart home approaches

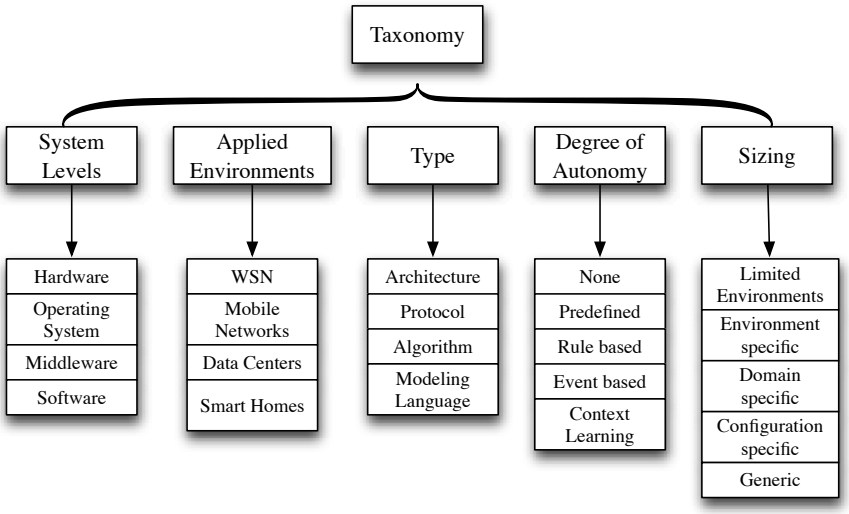

Figure 10. The comparison taxonomy.

System Levels

By this term, we refer to the level of the system where the energy adaptation takes place. We take into account the following system levels: Hardware, Operating System, Middleware, and Software. Most proposed solutions manage the software, hardware, or operating system levels in addition to the middleware level. Several solutions apply energy adaptations at the software level. Transhumance focuses on adapting the middleware platform, but the software is also adapted when the middleware platform adapts itself. GRACE applies multi-layer adaptations including per-application adaptations. Application-specific energy optimizations based on application classification are used in [16]. While CasCap adapts software and services in mobile devices. DYNAMO targets video streaming scenarios (*e.g.*, software) while trying to perform quality of service tradeoffs. The approach in [17] uses a dispatch algorithm to consolidate services in data centers. SANDMAN integrates several middleware platform techniques (middleware platform interfaces, service sessions or protocol choosing), however it also manages the hardware level by techniques such as switching devices' power mode or grouping devices in clusters based on similar mobility patterns. SleepServer targets also the hardware and software levels by managing energy through the creation of virtual images in order to keep the network availability of computers. The approach in [18] and GreenUp both target the hardware and software levels by managing the sleep of machines and their network presence through the usage of hardware and software proxies. GRACE

DOI: 10.1002/spe

and ECOSystem manage energy consumption and adaptations also on the operating system level. GRACE can apply adaptations at an OS level granularity, while ECOSystem is built on the *currentcy* model and used in OS scheduling and allocation. Other approaches presented in this review adapt the system mainly on the middleware level, without major energy oriented adaptations in other levels of the system.

Applied Environments

This refers to the main user environments that the proposed middleware platform approach targets. Environments range from *Wireless Sensor networks* (WSN) to other mobile networks, to large-scale systems, and to data centers. Because we limited our study to distributed environments, the approaches presented in this paper involve mobile networks or large-scale systems. Transhumance targets *Mobile Ad Hoc Networks* (MANETs). SANDMAN, on the other hand, targets larger environments: pervasive mobile networks and Wireless Sensor Networks (WSN). It seeks to reduce the energy consumption of idle devices in communicating networks. SleepServer, GreenUp and the approach in [18] are applied on computers in an enterprise environment (networked PCs, server availability). PARM also applies to mobile networks but its architecture and algorithmic approach are fit for large-scale systems. The approach in [17] is built for data centers. GRACE adopts a hierarchical approach for energy saving in different levels of computers (applications all system layers). The principle of the approach can be applied to distributed systems in general. DYNAMO's approach follows some of GRACE's ones. In particular the usage of different layer information to perform end-to-end adaptations. DYNAMO is targeted, however, to mobile devices. ECOSystem manages energy at the Operating System (OS) level, particularly managing computer tasks using a new energy abstraction mixing energy with currency: *currentcy*. CasCap and [16] targets applications in mobile devices running using network services or devices.

Type

This defines the type of the energy management approach. Types can be architectural, protocol, algorithm, or a modeling based. Architectural approaches are middleware platforms or frameworks that propose energy optimization techniques as architectural solutions Protocol approaches are where an energy efficient protocol is proposed in the middleware layer. Algorithmic approaches are where the latter is constructed around an energy-aware algorithm in the middleware layer. Modeling approaches are when a model is proposed for energy-awareness in the middleware layer. Most approaches presented in this review are architectural-based, with few exceptions: SANDMAN and PARM, in addition, propose protocol-based and algorithmic approaches, respectively. In SANDMAN, a self-adaptive discovery protocol is used as the basis of the energy management approach. PARM uses an algorithm to determine the components to migrate. This algorithm is crucial for the PARM framework. For the other approaches, [17] is a dispatch algorithm for data centers, and ECOSystem's approach is based on a new modeling definition: *currentcy*.

Degree of Autonomy

This indicates the degree of autonomy of the energy management approach (and not the overall degree of autonomy of the middleware platform itself). Autonomic computing refers to "*computing systems that can manage themselves given high-level objectives from administrators*" [28]. These systems are self-manageable where this *self-management* encompasses four main aspects: *self-configuration*, *self-optimization*, *self-healing*, and *self-protection*. They are formed by *autonomic elements*, which contain resources and deliver services. We choose several keyword expressions to describe the degree of autonomy. The expressions are: *i) None*, where the approach is not autonomous at all. This can be a system that applies energy adaptations through question/answer interactions with the user. *ii) Predefined*. In this case, the system uses predefined strategies, such as an algorithm, a protocol, static rules or a finite-state automaton. *iii) Rule-based*. Here, energy management adaptation is based on rules (or adaptive policies) that can be added and modified by an external entity (user, administrator). *iv) Event-based*. Systems that use an (complex) event

DOI: 10.1002/spe

processing engine falls in this category. The system collects data (context or energy data) and takes an adaptation decision as a result of events processing. And *v) Context Learning*. Here, energy management evolves by learning from context and energy information, and from the user habits. A level of artificial intelligence is also required for systems to be considered in this category.

Transhumance uses adaptation policies for middleware platform energy adaptations, and adaptation profiles for applications. Both techniques are based on a rule-based approach with conditions on the energy levels (local to one node or global to the network) and adaptation actions. Listing 1 presents an example of adaptation rules used in Transhumance. The solutions proposed in [16] and in [8] (CasCap) uses also adaptive policies for energy management. DYNAMO is also based on rules and on policies that can take the shape of utility factors. ECOSystem proposes a modeling definition, *currentcy*, and allocation and scheduling approaches that are predefined. PARM and [17] are based on algorithms that are defined prior to the execution of the system. SANDMAN uses an adaptable protocol and different predefined approaches for its energy awareness. SleepServer, GreenUp and [18] use a predefined approach allowing computers to go to sleep while preserving their network presence. GRACE is build around a multi-layer architecture. It is based on global and per-application algorithms to apply adaptations, and profiling for resource usage predictions, all of which are predefined prior to the execution of the system.

Sizing or Scalability

This describes the scale of the proposed approach. This means, how well can we apply the approach on different environments other than the ones that were presumably defined for. We use several keyword expressions for this taxonomy: *i) Limited Environment*, where the approach can be applied to an environment with very specific conditions. *ii) Environment-specific*. The approach is limited to one or few environments (such as approaches limited to WSN). This taxonomy is a subset of the wider domain-specific taxonomy. *iii) Domain-specific*. Here, the scope is wider than environment-specific. For example, an approach that can be applied on WSN and other mobile networks, fits in this taxonomy. *iv) Configuration-specific*, where the approach can be sized if a specific configuration is met, regardless of the applied environments or domains. And *v) Generic*. Here, the approach can be scaled on different domains, and thus can be considered as generic enough for a high degree of sizing.

Most of the reviewed approaches are environment-specific or domain-specific, with two more specific solutions: Transhumance, which requires additional preconditions, and PARM that has a wide domain scope but requires a specific configuration. Transhumance targets MANets with a maximum of 20 nodes and moving at a pedestrian speed (up to 5 km/h), which makes this a limited environment. ECOSystem emphasizes on the various computer tasks and networks. Thus, these approaches are domain-specific, same for SANDMAN, GRACE and [16]. SANDMAN targets pervasive networks in general, including WSN. SleepServer, GreenUp and [18] are specific to computer network within an enterprise environment (where servers are available to host virtual images for desktop PCs). The approach in [16], CasCap and DYNAMO are applied on mobile networks, while GRACE targets the very wide scope of distributed environments. Finally, PARM can be applied on different domains, from mobile networks to large-scale systems. However, it requires a specific configuration with the presence of proxies, brokers and servers.

Discussions

Based on our comparison we discuss how well the reviewed solutions fit in two main areas of research where contributions are still needed for a fully autonomous and easily sized energy-aware middleware platform.

Based on our comparison we discuss how well the reviewed solutions fit in an area of research where contributions are still needed for a fully autonomous middleware for energy management. Most of the solutions presented in this review follow two main approaches of autonomy: *rule-based* and *proxy-based* approaches, while the others use *predefined* techniques.

DOI: 10.1002/spe

Name	System Levels (+ Middleware)	Applied Environments	Type	Degree of Autonomy	Sizing (Scalability)
Transhumance	Software	MANets	Architecture	Rule-based	Limited Environment
SANDMAN	Hardware	Pervasive Mobile Networks, WSN	Architecture, Protocol	Predefined	Domain-specific
PARM	–	Mobile Networks, Large Scale Systems	Architecture, Algorithm	Predefined	Configuration-specific
GRACE/2	Operating System, Software	Distributed Systems	Architecture	Predefined	Domain-specific
Middleware for Energy-awareness in Mobile Devices	Software	Mobile Networks	Architecture	Rule-based	Domain-specific
Energy Consumption Optimized Service Hosting	Software	Data Centers	Algorithm	Predefined	Environment-specific
ECOSystem	Operating System	Computer Tasks	Modeling	Predefined	Domain-specific
CasCap	Software	Mobile Networks	Architecture	Rule-based	Domain-specific
DYNAMO	Software	Mobile Networks	Architecture	Rule-based	Domain-specific
SleepServer	Hardware, Software	Desktop PCs, Enterprise Networks	Architecture	Predefined	Domain-specific
Sleepless in Seattle No Longer	Hardware, Software	Desktop PCs, Enterprise Networks	Architecture	Predefined	Domain-specific
GreenUp	Hardware, Software	Desktop PCs, Enterprise Networks	Architecture	Predefined	Domain-specific

Table IV. Comparative table of middleware platform solutions for energy management

Prepared using speauth.cls
DOI: 10.1002/spe

Rule-based approaches offer an high degree of architectural autonomy, but with a limited decisional autonomy. The architecture of the middleware platform is flexible and evolutive, and can easily cope with changes in the environment. These architectures are based on autonomic control loop design [28], with subsystems designed to monitor, analyze, plan and execute energy-aware adaptations. Rules, on the other hand, need to be predefined and updated on environment's evolutions. None of the rule-based middleware platform approaches [3, 4, 5, 16, 8, 9] incorporate conflict management strategies. Transhumance [3, 4, 5], CasCap [8] and the approach in [16] all use *Event-Condition-Action* rules. The rule selection strategy is therefore simply a condition checked when an event occurs, and actions are applied regardless of their impact on the system or the conflicts that they may produce. The creation and the update process of these rules are left to the user, administrator or another application. The latters are responsible for verifying the coherence of the rules and the inexistance of rules conflicts and incompatibilites. DYNAMO [9] follows a similar process, although the usage of utility functions in complementary to ECA rules. This allows better modularity and flexibility in rule creation and evolution, but also allows the rule selection process to alleviate some of the conflicts. Conflicts may be prevented by using different rule selection strategies, such as fuzzy logic [29]. Or they can be dealt with a conflict management process based on system knowledge (through, for example, machine learning techniques), crowd-sourcing or modeling of the impact of applied actions. The distributed nature of the environment gives developers the opportunity to utilize the processing power and knowledge repositories that lies in the cloud. Finally, Transhumance is the only approach here where rule actions are applied on the middleware platform's modules only. The other approaches apply their rule actions on the managed system (software, system or hardware parameters).

Proxy-based approaches have the main advantage of keeping a device *available* for the user (or for other devices), while the device is in sleep mode. Therefore, this approach achieves both energy savings and limited disruption of the availability of the device. The reviewed proxy-based approaches [12, 13, 14, 18, 15], although sharing a similar proxy functioning, they differ in a key aspect : the proxy. In SANDMAN [12, 13], nodes regroup in clusters following similar criteria, then elect one of their own as a cluster head (or proxy). The cluster head is then responsible, not just for maintaining the presence of its nodes in the network, but also in deciding when a node is unused and thus can be put to sleep. On the contrary, the proxy in SleepServer [14], GreenUp [15] and in [18], are only responsible for keeping the availability of their sleeping machines in the network. They do not incorporate autonomic or intelligent functioning in relation to deciding when a machine should go to sleep. The nature of the proxy in the previous 3 approaches is different: *i)* a virtual machine image for SleepServer and unique to its original machine. Therefore, proxies are not shared as each machine have a unique proxy in the form of its virtual image. *ii)* an independent hardware proxy responsible for managing all machines in [14]. And *iii)* in GreenUp [18], a proxy is a normal machine that is elected using a distributed management technique. Although not similar, the selection process share some principles with SANDMAN's selection process.

The other approaches reviewed in this paper [6, 7, 10, 17, 11] have little autonomic functioning. They all use predefined techniques to manage energy, such as algorithmic adaptations and prediction algorithms (PARM [10], GRACE [6, 7] and [17]) or energy models (ECOSystem [11]).

Although these approaches offer a certain degree of autonomic management of energy consumption, a full autonomous energy management is yet to be defined. Autonomous context learning approaches for energy management at the middleware are missing. An energy management autonomous approach should therefore imitate the human body metabolism: the platform needs to be transparent to the user and to devices and applications, but without limiting users' high-level decisions. In the human body, when energy becomes low, the system starts by using its reserves and notifying the human about the situation (*e.g.*, the human feels hunger). Therefore, the human could apply high-level decisions, such as eating (to recharge his energy and reserves), or reduce his activity, or go to sleep (low power mode). We therefore believe that middleware platform approaches should take inspiration from biologic systems and provide a similar autonomous functioning for energy-awareness because the complexity of systems is rapidly increasing. Autonomous functioning means that users, in particular home users, do not need to do the job of a system administrator

DOI: 10.1002/spe

in managing the energy consumption of their software and devices. The absence of a system administrator, albeit for managing energy, is a challenge for ubiquitous computing, in particular in smart homes [30]. Energy, as a nun-functional requirement, needs more to be less disruptive for users than other business logic, thus the need for autonomous energy management.

In addition to autonomic functioning, all the reviewed approaches are specific to a specific domain or to a specific environment. Transhumance requires very specific conditions on the environment (small networks of up to 20 nodes and moving at pedestrian speed of up to 5 km/h). The platform focus mainly on adapting the middleware platform itself with limited adaptability for applications. On the contrary, PARM requires a specific configuration (the presence of proxies, brokers or directory services), but with a wider domain scope (*e.g.*, mobile networks, large scale systems). The algorithm presented in [17] can only be applied on data centers, thus is environment-specific. Similar to the previous approach, ECOSystem proposes a currency definition and a framework that targets the energy consumption at the OS level. SANDMAN targets energy consumption of idle devices in a communicating network, SleepServer, GreenUp and [18] manage energy consumption of idle devices in an enterprise network, while the architecture in [16] is valid for mobile networks. Finally, GRACE combines adaptations at different levels of the system in order to optimize the energy and resource utilization in a distributed environment. We believe that middleware platform approaches should allow some degree of scaling, not just inside a specific domain (such as the proxy-based approaches in enterprise networks [14, 18, 15]), but to a wider scope of applications. In the diversity of devices, networks and environments, an ideal middelware platform for energy management should be able to manage energy for multiple domains, ranging from mobile devices and desktop machines to distributed software in data center.

5. CONCLUSION

In this survey, we defined a comparative taxonomy for middleware approaches targeted at managing energy. We compared a number of existing approaches and identified two major research fields where contributions are lacking for energy management middleware: *autonomic approaches* and *generic approaches* for energy management middleware platforms.

Autonomic approaches are a crucial step in order to cope with the widespread usage of software services and the multiplication of available digital devices. The complexity of these distributed and pervasive environments and the diversity of its energy requirements and characteristics can only be efficiently challenged by autonomic solutions. The reviewed rule-based solutions offer some degree of autonomous functioning but they lack autonomous management (creation, evolution) of rules and applied actions. They also lack the integration of more coherent and non-conflicting selection processes. Proxy-based approaches either use predefined protocols for energy management, or leave the intelligence part up to the user.

Genericity is also an important aspect, although less crucial compared to autonomic model. The fast evolution of technologies and energy specifications in hardware and software, implies that non-adaptable or context-specific approaches are quickly becoming outdated. Generic energy solutions or models should therefore set the basics of evolving architectures that can cope with these challenges.

Finally, the impact on user experience of the energy management platform (*i.e.*, energy measurements, analysis, and adaptations) should be reduced. Ideally, the middleware platform for energy management should be invisible, both to the user (in terms of impact of usability), to the system (in terms of impact on computational time and resources utilization), and to the energy domain itself (in term of its own energy consumption).

Energy management middleware approaches is an ongoing research, and is becoming more crucial as energy is a limited and scarce resource. Developing autonomic middleware platforms and software architecture for energy-awareness, with levels of genericity, is therefore an open and important research area.

DOI: 10.1002/spe

FUNDING

This work is partially funded by the French Ministry of Higher Education and Research, Nord-Pas de Calais Regional Council and FEDER through the *Contrat de Projets Etat Region Campus Intelligence Ambiante* (CPER-CIA) 2007-2013.

REFERENCES

1. Krakowiak S. *Middleware Architecture with Patterns and Frameworks*. Sacha Krakowiak, http://sardes.inrialpes.fr/~krakowia/MW-Book/, 2007.
2. Lu YH, Qiu Q, Butt AR, Cameron KW. End-to-end energy management. *Computer* 2011; **44**:75–77, doi: 10.1109/MC.2011.342.
3. Paroux G, Martin L, Nowalczyk J, Demeure I. Transhumance: A powersensitive middleware for data sharing on mobile ad hoc networks. *Proceedings of the 7th international Workshop on Applications and Services in Wireless Networks (ASWN'07)*, Spain, 2007.
4. Paroux G, lsabelle Demeure, Reynaud L. A Power-aware Middleware for Mobile Ad-hoc Networks. *Proceedings of the 8th International Conference on New Technologies in Distributed Systems (NOTERE'08)*, ACM, 2008; 1–7, doi:10.1145/1416729.1416757.
5. Demeure I, Paroux G, Hernando-ureta J, Khakpour AR, Nowalczyk J. An energy-aware middleware for collaboration on small scale manets. *Proceedings of the Autonomous and Spontaneous Networks Symposium (ASN'08)*, Paris, France, 2008.
6. Sachs DG, Yuan W, Hughes CJ, Harris A, Adve SV, Jones DL, Kravets RH, Nahrstedt K. Grace: A hierarchical adaptation framework for saving energy 2004.
7. Vardhan V, Yuan W, III AFH, Adve SV, Kravets R, Nahrstedt K, Sachs DG, Jones DL. Grace-2: integrating fine-grained application adaptation with global adaptation for saving energy. *International Journal of Engineering Science* 2009; **4**(2):152–169, doi:10.1504/IJES.2009.027939.
8. Xiao Y, Hui P, Savolainen P, Ylä-Jääski A. Cascap: cloud-assisted context-aware power management for mobile devices. *Proceedings of the 2nd international workshop on Mobile cloud computing and services*, MCS '11, ACM: New York, NY, USA, 2011; 13–18, doi:10.1145/1999732.1999736.
9. Mohapatra S, Dutt N, Nicolau A, Venkatasubramanian N. Dynamo: A cross-layer framework for end-to-end qos and energy optimization in mobile handheld devices. *Selected Areas in Communications, IEEE Journal on* may 2007; **25**(4):722 –737, doi:10.1109/JSAC.2007.070509.
10. Mohapatra S, Venkatasubramanian N. Parm: Power aware reconfigurable middleware. *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS'03)*, IEEE Computer Society: Washington, DC, USA, 2003; 312.
11. Zeng H, Ellis CS, Lebeck AR. Experiences in managing energy with ecosystem. *IEEE Pervasive Computing* January 2005; **4**:62–68, doi:10.1109/MPRV.2005.10.
12. Schiele G, Becker C. SANDMAN: an Energy-Efficient Middleware for Pervasive Computing. *Technical Report*, University of Mannheim, Karlsruhe, Germany october 2007.
13. Schiele G, Handte M, Becker C. Experiences in designing an energy-aware middleware for pervasive computing. *Proceedings of the 6th Annual IEEE International Conference on Pervasive Computing and Communications (PERCOM'08)*, IEEE Computer Society: Washington, DC, USA, 2008; 504–508, doi:10.1109/PERCOM.2008.92.
14. Agarwal Y, Savage S, Gupta R. Sleepserver: a software-only approach for reducing the energy consumption of pcs within enterprise environments. *Proceedings of the 2010 conference on USENIX annual technical conference*, USENIX Association: Berkeley, CA, USA, 2010; 22–22.
15. Sen SS, Lorch JR, Hughes R, Garcia Jurado Suarez C, Zill B, Cordeiro W, Padhye J. Dont lose sleep over availability: The greenup decentralized wakeup service. *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI'12)*, 2012.
16. Xiao Y, Kalyanaraman RS, Ylä-Jääski A. Middleware for energy-awareness in mobile devices. *Proceedings of the 4th International ICST Conference on Communication System software and middleware (COMSWARE'09)*, ACM: New York, NY, USA, 2009; 1–6, doi:10.1145/1621890.1621907.
17. Binder W, Suri N. Green Computing: Energy Consumption Optimized Service Hosting. *Proceedings of the 35th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '09)*, Springer-Verlag: Berlin, Heidelberg, 2009; 117–128, doi:10.1007/978-3-540-95891-8-14.
18. Reich J, Goraczko M, Kansal A, Padhye J. Sleepless in seattle no longer. *Proceedings of the 2010 conference on USENIX annual technical conference*, USENIX Association: Berkeley, CA, USA, 2010; 17–17.
19. Becker C, Schiele G, Gubbels H, Rothermel K. Base - a micro-broker-based middleware for pervasive computing. *Proceedings of the 1st IEEE International Conference on Pervasive Computing and Communications, 2003 (PerCom'03)*, 2003; 443–451, doi:10.1109/PERCOM.2003.1192769.
20. AMD. Magic Packet Technology. *Technical Report*, AMD White Paper 1995.
21. Capone A, Barros M, Hrasnica H, Tompros S. A new architecture for reduction of energy consumption of home appliances. *Towards eEnvironment, European conference of the Czech Presidency of the Council of the EU (e-Envi'2009)*, Prague, Czech Republic, 2009.
22. The AIM consortium. http://www.ict-aim.eu.
23. Eisenhauer M, Rosengren P, Antolin P. A development platform for integrating wireless devices and sensors into ambient intelligence systems. *Proceedings of the 6th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops (SECON'09).*, 2009; 1–3, doi:10.1109/SAHCNW.2009.5172913.

24. Jahn M, Jentsch M, Prause C, Pramudianto F, Al-Akkad A, Reiners R. The energy aware smart home. *Proceedings of the 5th International Conference on Future Information Technology (FutureTech'10)*, 2010; 1–8, doi:10.1109/FUTURETECH.2010.5482712.
25. The Hydra Middleware. http://www.hydramiddleware.eu.
26. Mike M H, Evan M R, Xiaofan J, Randy HK, Seth R S, David E C, Ken L. An Architecture for Local Energy Generation, Distribution, and Sharing. *Proceedings of the IEEE Energy 2030 Conference*, IEEE, 2008; 1–6, doi:10.1109/ENERGY.2008.4781028.
27. The LoCal research team. http://local.cs.berkeley.edu.
28. Kephart JO, Chess DM. The Vision of Autonomic Computing. *IEEE Computer* 2003; **36**(1):41–50, doi:10.1109/MC.2003.1160055.
29. Zadeh L. Fuzzy sets. *Information and Control* 1965; **8**(3):338 – 353, doi:10.1016/S0019-9958(65)90241-X.
30. Edwards WK, Grinter RE. At home with ubiquitous computing: Seven challenges. *Proceedings of the 3rd international conference on Ubiquitous Computing*, UbiComp '01, Springer-Verlag: London, UK, UK, 2001; 256–272.

DOI: 10.1002/spe